CACHE MANAGEMENT METHOD AND SYSTEM FOR STORING DYNAMIC CONTENTS

Background of the Invention

1. Field of the Invention

The present invention relates to cache management and, more particularly, to a method and system for managing a cache of a server based on the levels of resources used to create dynamic content stored in the cache.

2. Description of the Related Art

Web pages or other computer pages that are created using variable data are known as "dynamic" pages. Generally, an application server on a network creates these dynamic pages by running programs such as Servlets. Each time a dynamic page is created, however, a significant amount of resources are used. Among the resources used, most page providers (e.g., website providers) are concerned with the monetary cost involved with generating a dynamic page. The generation cost of a dynamic page typically includes, but is not limited to, the cost of running the CPU (Central Processing Unit) on the application server, the cost of invoking EJB (Enterprise Java Bean) and JSP (Java Servlet Page) modules typically used to generate HTML file(s) corresponding to the page, and the network cost of invoking the application server and other processing devices. In most

systems, the generation cost for a dynamic page can be rather high, which cost the page provider bears each time a dynamic page is created.

To reduce the costs associated with providing dynamic pages to users, it is common practice to store in a cache located on the application server some of the previously created dynamic pages that are frequently requested by the users. A cache is a temporary storage unit or location for temporarily storing previously created dynamic content such as dynamic pages, so that the application server need not recreate the same page each time a user requests it. However, not all previously created pages can be stored in the cache of the application server because the cache has limited memory space. To address this problem, certain criteria or page replacement methods may be used to determine which of the dynamically created pages should be "saved" in the cache for a subsequent use.

Several page replacement methods are known for managing the cache of an application server in a network environment. Among these methods, most frequently used are the Least Recently Used (LRU) method and the Hit Count method. In the LRU method, the system is configured to replace the least recently used pages stored in the cache with newly created pages. That is, the criteria used for replacing the cached pages is the time and date of the most recent access of the pages currently stored in the cache. In the Hit Count method, the system is configured to replace pages that are accessed less frequently. In other words, the cached pages are replaced based on how frequently the cached pages are accessed. These replacement methods, however, fail to consider the amount of resources used in, or page generation costs associated with, creating the

cached pages. As a result, the prior art systems may replace a cached page with a new page even though it may be more expensive to recreate the cached page than to recreate the new page. Thus, the conventional cache management methods do not serve effectively the needs of page providers.

With the advent of personalized web pages and the growing number of business websites found on the web, the number of dynamic pages that are temporally stored or "cached" is growing at a fast rate. As the number of dynamic pages requiring complex back-end processing grows, the cache management methods for dynamic content become much more important because they can have a significant impact on the overall operation of the system. It is thus important to have a cache management system and method which can overcome the problems of conventional cache management methods and which can reduce the page provider's costs of providing dynamic pages to end-users.

Summary of the Invention

The present invention provides an improved cache management method and system for storing dynamic contents in a computing environment. The cache management method considers predetermined parameters associated with the dynamic content, such as the costs associated with generating web pages, and selectively replaces the dynamic contents that are stored in the cache with new dynamic content based on the predetermined parameters, e.g., the generation costs. In this manner, storage of dynamic content in the cache is prioritized based on the predetermined parameters. In the previously described web page example, pages stored in the cache that are expensive to

recreate are not replaced with the new pages and the costs of providing dynamic pages to the users can be reduced significantly.

In a preferred embodiment, each time a dynamic page is created, the generation cost information identifying the cost of generating that page is associated with that page, e.g., using tags . A newly generated page can be stored in the cache of a server if there is an empty slot in the cache, so that the new page can be retrieved from the cache for a subsequent use. If the cache is full, however, the server uses the generation cost information to determine if any of the cached pages can be replaced with the new page. The server compares the generation cost of the new page with the generation costs of the cached pages and selects a cached page that is less expensive to recreate compared to the new page. The selected cached page is replaced with the new page in the cache.

Thus, the present invention provides a cache management system and method which implements a page replacement technique having a more global view, in that it considers the entire cost of generating a dynamic page from all involved machines and networks before the page can be replaced in the cache. As a result, the cost of providing dynamic pages to end-users can be reduced significantly and the efficient allocation of resources can be achieved in connection with the generation of dynamic pages.

Brief Description of the Drawings

Figure 1 is a block diagram of a computer workstation environment in which the present invention may be practiced.

Figure 2 is a diagram of a networked computing environment in which the present invention may be practiced.

Figure 3 illustrates a flowchart showing the processing steps involved in a cache management method according to a simple embodiment of the present invention.

Figure 4 illustrates a flowchart showing the processing steps involved in Step 215 of Fig. 3 according to the present invention.

Figure 5 is an example of a table which may be used in the present invention.

Figure 6 illustrates a flowchart showing the processing steps involved in a cache management method which may be implemented in a network environment according to another embodiment of the present invention.

Detailed Description of the Preferred Embodiments

Fig. 1 illustrates a representative workstation hardware environment in which the present invention may be practiced. The environment of Fig. 1 comprises a representative single user computer workstation 10, such as a personal computer, including related peripheral devices. The workstation 10 includes a microprocessor 12 and a bus 14 employed to connect and enable communication between the microprocessor 12 and the components of the workstation 10 in accordance with known techniques. The workstation 10 typically includes a user interface adapter 16, which connects the microprocessor 12 via the bus 14 to one or more interface devices, such as a keyboard 18, a mouse 20, and/or other interface devices 22, which can be any user interface device, such as a touch sensitive screen, a digitized entry pad, etc. The

bus 14 also connects a display device 24, such as an LCD screen or monitor, to the microprocessor 12 via a display adapter 26. The bus 14 also connects the microprocessor 12 to memory 28 and long-term storage 30 which can include a hard drive, diskette drive, tape drive, etc.

The workstation 10 may communicate with other computers or networks of computers, for example via a communications channel or modem 32. Alternatively, the workstation 10 may communicate using a wireless interface at 32, such as a CDPD (Cellular Digital Packet Data) card. The workstation 10 may be associated with such other computers in a local area network (LAN) or a wide area network (WAN), or the workstation 10 can be a client in a client/server arrangement with another computer, etc. All of these configurations, as well as the appropriate communications hardware and software, are known in the art.

Fig. 2 illustrates a data processing network 40 in which the present invention may be practiced. The data processing network 40 may include a plurality of individual networks, such as wireless network 42 and network 44, each of which may include a plurality of individual workstations 10. Additionally, as those skilled in the art will appreciate, one or more LANs may be included (not shown), where a LAN may comprise a plurality of intelligent workstations coupled to a host processor.

Still referring to Fig. 2, the networks 42 and 44 may also include mainframe computers or servers, such as a proxy server 46 and/or application server 47 (which may access a data repository 48). The proxy server 46 serves as a point of entry into each network 42, 44 and may also be known as an "edge server" or "gateway

computer." The proxy server 46 is an intermediary server which interfaces between the application server 47 and the workstations 10, and may maintain its own cache for temporarily storing or "caching" dynamic contents, e.g., dynamic pages. The proxy server 46 may be coupled to another network 42 or 44 by means of a communications link 50a and communicate with an application server in that network. The proxy server 46 may also be directly coupled to one or more workstations 10 using a communications link 50b, 50c. The proxy server 46 may also be coupled 49 to a storage device (such as data repository 48 or other data repositories). Further, the proxy server 46 may be indirectly coupled to one or more workstations 10. The proxy server 46 relays requests and information between the application server 47 and the workstations 10. If the proxy server 46 is unable to process a request (e.g., a page request) transmitted from the workstations 10, it relays the request to an appropriate application server 47 which then responds to the request. Here, the application server 47 may be located in the same network or in another network remotely located from the proxy server 46.

The application server 47 maintains its own cache for temporarily storing the dynamic pages. In response to a page request from the proxy server 46, the application server 47 invokes programs (e.g., Servlets) to create the requested page if the requested page is not available from the cache of the application server 47 and transmits the created page to the proxy server 46. However, if the requested page is stored in the cache of the application server 47, the application server 47 merely retrieves it from the cache and transmits the retrieved page to the proxy server 46. The

proxy server 46 then redirects the page back to the requester's workstation 10. A main difference between the application server 47 and the proxy server 46 is that the application server 47 can create dynamic pages if needed, whereas the proxy server 46 cannot.

The workstations 10 may be connected to the wireless network 42 using a networking protocol such as the Transmission Control Protocol/Internet Protocol ("TCP/IP") over a number of alternative connection media, such as cellular phone, radio frequency networks, satellite networks, etc. The wireless network 42 preferably connects to the proxy server 46 using a network connection 50a such as TCP or UDP (User Datagram Protocol) over IP, X.25, Frame Relay, ISDN (Integrated Services Digital Network), PSTN (Public Switched Telephone Network), etc. The workstations 10 may alternatively connect directly to the proxy server 46 using dial connections 50b or 50c. Further, the wireless network 42 and network 44 may connect to one or more other networks (not shown), in an analogous manner to that depicted in Fig. 2. Moreover, the application server 47 may be located a great distance from the proxy server 46 and may communicate with the proxy server 46 via a network.

For the purposes of illustration only and not of limitation, a preferred embodiment of the present invention is now described herein in terms of servicing a request for a dynamic page to be delivered to a requesting client. However, the principles of the present invention are equally applicable to servicing requests for any dynamic contents or data. The requester of page content may, in some cases, be application software. Thus the references herein to a client are intended to include

such scenarios, in addition to those in which the requester is an end-user. Furthermore, the term "computer page" or "page" is used herein for ease of reference only and is intended to cover any form of a dynamic web page.

Fig. 3 is a flowchart of the processing steps involved in a cache management method according to a simple embodiment of the present invention. These steps may be implemented by the application server 46 discussed in connection with Fig. 2. As shown in Fig. 3, a dynamic page is generated in Step 200 responsive to a page request from a user or from another server such as a proxy server. The page can be generated using known page generation techniques, e.g., by running page generation programs such as Servlets. At this or later time, the generated page is delivered to the page requester.

In Step 205, the cost of generating the created page is determined. Various of methods of determining the generation cost for a dynamic page are known and can be use herein. For example, the cost of generating a dynamic page can be calculated manually, or the system may be configured to calculate the cost using certain computer programs or algorithms. Once the generation cost information is obtained, this information is associated with the created page in Step 210. In accordance with an exemplary embodiment, one or more "tags" are used to associate the generation cost information with the page. The use of tags to associate data is well known in the art. In the present invention, the tags themselves may carry the generation cost information or may function merely as a pointer indicating where the generation cost information can be found. The tag(s) may be coupled to the page file directly or to the identifier of

the page, e.g., the URL (Uniform Resource Locator) corresponding to the page. Once the generation cost information is associated with the page, a caching process for selectively storing the page in the cache of a server according to the present invention is performed in Step 215.

Fig. 4 is a flowchart illustrating the processing steps involved in Step 215 of Fig. 3. As shown in Fig. 4, in Step 400, a determination is made whether the cache of the server is full. If the cache is not full, then in Step 405, the page is stored in the cache of the server along with the tag(s) identifying the generation cost information. In this manner, each of the pages stored in the cache is associated with the generation cost information identifying the generation cost of that page. If in Step 400 the determination indicates that the cache is full, the generation cost tag(s) associated with the current page are examined in Step 410. Then in Step 420, the generation cost information associated with each of the pages stored in the cache is examined to search for a cached entry having the page generation cost that is lower than the generation cost of the current page. If the search indicates in Step 425 that a cached entry with a lower generation cost is not found, then in Step 430 it is determined that the current page cannot be added to the cache and the caching process ends. The pages that are not cached may be discarded after the pages are returned to the page requester. If, in Step 425, the search indicates that a cached entry with a lower generation cost is found, the lower-cost cached entry is replaced with the current page and the associated cost tag(s) are stored in Step 435. If the search locates multiple cached entries with lower generation costs, the cached entry with the lowest generation

cost among the located cached entries may be replaced, or other schemes may be utilized to narrow the entries to one. For instance, applying the conventional Least Recently Used (LRU) method or Hit Count method herein, from the multiple cached entries that are found, the entry that was least recently or least frequently accessed by the user may be replaced with the current page and its tags. After Step 435, the caching processing of Fig. 4 is completed.

Fig. 5 shows one example of a table which may be used in the search process of Step 420 in Fig. 4 according to the present invention. As shown in Fig. 5, the table identifies (1) a series of cache entry numbers 230 (or addresses) each identifying a particular location in the cache where a page is stored, (2) a series of page identifiers (e.g., URLs) 220 each identifying the page stored in the associated cache entry location, and (3) a list of the generation cost tags 225 each associated with a particular page identifier. In utilizing the table in the search process of Step 420, for example, if the generation cost of the currently generated page is 6 and the system is configured to recognize that lower tag values indicate lower generation costs, then a hashing technique or other search method may be used to search the table and determine whether any of the generation cost tags has a value less than 6. In this example, the cache entry number 1 has the generation cost tag value of 5 and is thus selected as having the generation cost lower than the generation cost of the current page. Accordingly, the page stored at the cache entry number 1 location of the cache and identified by URL10 is replaced with the current page. Thereafter, the table is updated to reflect this replacement.

Fig. 6 depicts the processing steps involved in a cache management method according to another embodiment of the present invention. These steps may be implemented in the computer network environment illustrated in Fig. 2 such that, in one embodiment, a client 300, a proxy server 305, and an application server 310 in Fig. 6, all operatively coupled, may represent respectively the workstation 10, the proxy server 46, and the application server 46 in Fig. 2. In the alternative, these processing steps may be practiced in other computing environments or in other components. In the embodiment shown in Fig. 6, any page stored in the cache of a server is associated with the generation cost information identifying the cost of generating that page, e.g., using tags, as discussed hereinabove.

Referring now to Fig. 6, in Step 320, a page or content request is generated by the client 300, e.g., by a user entering a URL of the desired page/website in his computer or workstation 10. The page request is sent to the network (e.g., under control of the user's web browser) and received by the proxy server 305 in Step 340. In some applications, the proxy server 305 may be located at the client's Internet Service Provider (ISP) site. If the proxy server 305 is configured to maintain its own cache, then the proxy server 305 determines in Step 345 whether the requested page is available from its cache. Those skilled in the art would appreciate that the proxy server 305 is not required to have its own cache. Instead, it may receive caching services through some other means such as a stand alone cache or an external cache and Step 345 is equally applicable in such situations.

If the requested page is available from the cache (or caching services) of the proxy server 305 (i.e., Step 345 has a positive result), the proxy server 305 retrieves that page from the cache and returns it to the client 300 in Step 355, and the page replacement process of the present invention is not invoked to provide the requested page. Upon receipt of the returned page in Step 325, the client 300 (i.e., the computer) either displays the page or further processes it in Step 330 using techniques that are well known in the art. The processing of the page request from the client 300 is then completed.

If the requested page is not available from the cache of the proxy server 305 (i.e. Step 345 has a negative result), then the proxy server 305 forwards the page request to an appropriate application server 310 in Step 350. In some cases, the application server 310 may be a server located at the main terminal servicing the requested page or related website. For instance, a user's request for an IBM web page having the URL "www.ibm.com" may be processed by an application server such as the IBM main frame server servicing IBM websites.

Upon receipt of the forwarded page request at Step 375, the application server 310 searches its own cache for the requested page at Step 380. If the requested page is found in the cache of the application server 310, the application server 310 retrieves the requested page from its cache and returns it to the proxy server 305 in Step 385.

When the determination in Step 380 indicates that the requested page is not available from the cache of the application server 310, the process proceeds to Step 390 where the application server 310 creates the requested page and the cost of

generating the requested page is determined and associated with the page, e.g., using tags. In other words, Step 390 is identical to the combination of Steps 200, 205 and 210 in Fig. 3. Then Step 215, previously described in connection with Fig. 4 above, is performed wherein the created page is evaluated for storage in the cache of the application server 310 using the generation cost tags associated with the pages stored in the cache of the application server 310. After the page has been cached (or, alternatively, prior to or contemporaneously with caching the page), the application server 310 returns the created page and its generation cost tags to the proxy server 305 in Step 385.

The proxy server 305 receives in Step 360 the requested page and its tags from the application server 310. Then Step 215 (see Fig. 4) is performed wherein the proxy server 305 evaluates whether the received page should be stored in the cache of the proxy server 305 for a subsequent use. After Step 215, the received page is returned to the requesting client 300 in Step 370. From Step 370, the process proceeds to Steps 325 and 330 where the page is processed as previously described hereinabove. The processing of the page request from the client 300 is then completed according to this embodiment.

Those skilled in the art would readily understand that the order of certain steps shown in Fig. 6 may be changed without affecting the outcome according to the present invention. For example, the order of Steps 215 and 385 (performed by the application server) may be reversed, or these steps may occur simultaneously. Similarly, the order of Steps 215 and 370 (performed by the proxy server) may be reversed, or these steps

may occur simultaneously. Furthermore, in Steps 345 and 380, the requested page may not become available from the cache for accessing by the server even when the page is stored in the cache, if the cached page has become stale or invalid according to prior art invalidation criteria. In these cases, the determination made in Step 345 or 375 would yield a negative result.

In some applications, the proxy server 305 does not maintain its own cache. If so, Steps 345, 355, and 365 are omitted (e.g., proceed from Step 340 directly to Step 350; from Step 360 directly to 370). In addition, it may happen that more than one intermediate device which provides caching, such as the proxy server 305, is encountered in the network path. In this case, each such intermediate device may perform the logic depicted in Steps 340 through 370 including Step 215.

In accordance with the present invention, the tags associated with each content may identify content generation costs, so that those cached contents that are more expensive to recreate than to recreate the newly created content, are not replaced and can be maintained in the cache of a server. The generation costs for each page may include, but are not limited to, the cost of running the CPU on the application server, the cost of invoking EJB (Enterprise Java Bean) and JSP (Java Servlet Page) modules typically used to generate HTML or other files corresponding to the page, and the network cost of invoking the application server and other processing devices. It should be clearly understood that, although the tags are herein described in association with the generation costs, the tags in accordance with the present invention may identify any other parameters or additional resource information associated with the generation

of a page, e.g., a duration of time involved in generating a page. The tags identifying other resource information can be used as an alternative or in conjunction with the tags identifying the generation costs. In some embodiments, default tags may be used when pages with faulty tags are found or the tags themselves cannot be located. A default tag may represent the page as the "lowest priority" item or as highly replaceable in order to provide compatibility with caching systems and service providers that do not support the generation cost levels of the present invention.

In still other embodiments, the conventional replacement policies such as the LRU or Hit Count algorithms can be further combined with the resource-based replacement policy of the present invention. For instance, when a time arises to determine if the newly created content should replace any entry stored in the cache of a server, the cache may be searched for an entry that is least recently or least frequently accessed by the user as in the LRU or Hit Count method. If the search locates such an entry, the tag(s) associated with that entry identifying the resource information can be examined. If the resource tag(s) associated with the located entry indicate that the regeneration of the located entry requires more resources or higher generation costs than the newly created content, then the located entry may not replaced by the new content. This results in a page replacement policy for a cache management system which maximizes an efficient allocation of resources for the system.

The present invention may be implemented in a number of distributed computing environments. For example, the implementation of the present invention may occur in

a web environment, where an application/web server provides services in response to requests transmitted using the HyperText Transfer Protocol (HTTP) from a client connected through the Internet. Alternatively, the implementation of the present invention may be realized in a non-web environment (e.g., using the Internet, a corporate intranet or extranet, or any other network) where cached information is accessed by distributed applications (e.g., using techniques such as: Remote Method Invocation or RMI; IIOP which is the Internet Inter-ORB Protocol; etc). Configurations for the environments in which the present invention may be practiced include a client/server network, as well as a multi-tier environment. These environments and configurations are well known in the art.

A user of the present invention may connect his computer to a server using a wireline connection, or a wireless connection. Wireline connections are those that use physical media such as cables and telephone lines, whereas wireless connections use media such as satellite links, radio frequency waves, and infrared waves. Many connection techniques can be used with these various media, such as: using the computer's modem to establish a connection over a telephone line; using a LAN card such as Token Ring or Ethernet; using a cellular modem to establish a wireless connection; etc. The user's computer may be any type of computer processor, including laptop, handheld or mobile computers; vehicle-mounted devices; desktop computers; mainframe computers; etc., having processing and communication capabilities. The remote server, similarly, can be one of any number of different types of computer which have processing and communication capabilities. These techniques

are well known in the art, and the hardware devices and software which enable their use are readily available.

The application server or proxy server of the present invention may be implemented utilizing an Enterprise Systems Architecture/370 available from IBM, an Enterprise Systems Architecture/390 computer, etc. Depending on the application, a midrange computer, such as an Application System/400 (also known as an AS/400) may be employed. ("Enterprise Systems Architecture/370" is a trademark of IBM; "Enterprise Systems Architecture/390", "Application System/400", and "AS/400" are registered trademarks of IBM.) The application server or proxy server may also be RS/6000, Netfinity, any PC or Unix machine, or any server hardware capable of hosting a web proxy server or web application server.

Software programming code which embodies the present invention is typically accessed by the microprocessor of the application server or proxy server from long-term storage media of some type, such as a CD-ROM drive or hard drive. The software programming code may be embodied on any of a variety of known media for use with a data processing system, such as a diskette, hard drive, or CD-ROM. The code may be distributed on such media, or may be distributed from the memory or storage of one computer system over a network of some type to other computer systems for use by such other systems. Alternatively, the programming code may be embodied in memory, and accessed by the microprocessor of the application server or proxy server using the bus in the server. The techniques and methods for embodying software programming

code in memory, on physical media, and/or distributing software code via networks are well known and will not be further discussed herein.

The present invention may be implemented as one or more modules (also referred to as code subroutines, or "objects" in object-oriented programming) of one or more computer software programs. Preferably, this software operates on a server (such as an application server) or intermediary device (such as an edge or proxy server) in a network. Or, the software may execute on multiple devices in a distributed manner. For example, multiple application servers may share cached data in a centralized database, and each such application server may implement the selective page replacement techniques of the present invention. The present invention may also be used with cached (or cachable) content that is structured along any boundary, such as a fragment of a document or page. .

Thus, the present invention provides a novel technique for selectively replacing cached contents and is applicable to distributed computing environments such as client-server environments or other network environments. The selective page replacement techniques of the present invention provide enhanced return services to clients wherein the client's cached (or cachable) content is given prioritized treatment based on resources (e.g., generation costs) needed to recreate the requested content. In this manner, the requested content (e.g., web pages) can be returned to the client more quickly with the minimum use of system resources. Web page providers (e.g., e-merchants, ISPs, etc.) may offer the enhanced return service to their clients for free, for a fixed fee, or for a fee varying according to a graduated pricing policy. Statistics may

be accumulated to justify the fee, e.g., by accumulating a count of the number of times Step 435 in Fig. 4 is executed for each user, which caused the user's requested content to take priority over other already-cached contents. Or, conversely, a count may be accumulated of the number of times the user's cached content was "protected" from being overwritten in the cache by newly created content when the test in Step 425 in Fig. 4 outputs a negative result.

With the advent of the J2EE programming model and the amount of enterprise data being provided to web applications from back-end servers, it is desirable to understand the impact of the page replacement policy on the computing resources that supply the pages. By providing a page replacement policy that relies on such computing resources, the present invention provides the ability to better manage the costs associated with providing and maintaining sites on the network. It allows providers of web hosting sites and pages to maintain the lowest possible end-to-end costs for their sites and pages.

Although the present invention has been described with respect to a specific preferred embodiment thereof, various changes and modifications may be suggested to one skilled in the art and it is intended that the present invention encompass such changes and modifications as they fall within the scope of the appended claims.